On-Demand TDMA Scheduling for Energy Conservation in Sensor Networks

Technical Report

June 2004

Matthew J. Miller
Department of Computer Science, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
mjmille2@uiuc.edu

Nitin H. Vaidya
Department of Electrical and Computer Engineering, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
nhv@uiuc.edu

***Abstract*— In this paper, we present two variants of an on-demand TDMA (Time Division Multiple Access) MAC protocol. These protocols are designed specifically for sensor networks. Thus, they attempt to reduce energy consumption while still providing efficient delivery of sensor data to the sinks. The two variants of the protocol presented in this paper are Busy Tone On-Demand Scheduling (BTODS) and On-Demand Scheduling (ODS). BTODS is designed for sensors capable of using non-interfering channels to transmit busy tones, while ODS is designed for single channel architectures. These protocols are designed to schedule slots for sensor-to-sink flows that do not interfere with existing flows. Thus, nodes can conserve energy by switching to a sleep state in slots in which they are neither sending nor receiving data. The tradeoffs between the two protocols are discussed in detail, along with the protocols' interactions with upper layers in the network stack.**

I. Introduction

The emergence of sensor networks presents many new challenges in wireless ad-hoc networks. While the precise application of sensor networks is speculative, certain characteristics are typically assumed. First, the sensors are static after initial deployment (unless placed on a mobile entity [1]). Second, energy is scarce and it is inconvenient or impossible to replenish the energy source frequently.

Because energy should be conserved, power save protocols are needed. At the MAC layer, we seek to switch the wireless radio off as much as possible without significantly hindering performance metrics such as latency and throughput. The four major sources of energy waste at the MAC layer are [2]:

Collisions When packets collide, energy is wasted because the packet, along with associated control overhead, must be retransmitted.

Overhearing This occurs when a node promiscuously listens to a packet intended for a different destination. In this case, the node could sleep rather than idly listening to the channel.

Control Overhead Aside from the data packet, which has additional header bytes prepended, usually other MAC level packets add overhead. For example, an ACK packet is usually required from the receiver to verify whether the data transmission was a success or failure.

Idle Listening This is when a node is listening to the channel, but not transmitting or receiving any data. This typically consumes much more energy than if the node were to sleep.

This work is supported in part by NSF grant ANI-0125859 and a NDSEG fellowship.

TABLE I
Characteristics of a sensor radio [3].

Radio State	Power Consumption (mW)
Transmit	81
Receive/Idle	30
Sleep	0.003

Radios typically have four power levels corresponding to the following states: transmitting, receiving, listening, and sleeping. Typically, the power required to listen is about the same as the power to transmit and receive. The sleep power is usually one to four orders of magnitude less. Thus, idle listening is the largest source of waste in sensor networks. A sensor should sleep as much as possible when it is not engaged in communication. The power levels for Mica Mote sensors [3] are shown in Table I.

In this work, we propose a Time Division Multiple Access (TDMA) protocol to schedule flows of sensor traffic in non-interfering slots. Scheduling the flows can reduce the energy wasted from the aforementioned sources. In a steady-state, collisions should not occur because each flow has chosen a slot which does not interfere with its neighbors' communications. Overhearing and idle listening are less of a problem because sensors only need to wakeup in slots in which they are scheduled to send or receive. It is difficult to quantify the effects of overhearing avoidance on overall energy consumption because some upper-layer protocols [4], [5] use promiscuous listening to improve performance. We do not explicitly address this, but note that nodes can choose to listen in other slots if this is desired.

There are some disadvantages of a TDMA protocol when compared to a random access protocol, like IEEE 802.11 [6]. First, scheduling must be done by predicting future traffic arrivals, delaying packets until scheduling can be done, or requiring traffic to follow a predictable pattern. Also, TDMA requires synchronization to divide time into slots across the whole network. However, as mentioned in Section II, we believe sensor networks can be designed to overcome these disadvantages.

In addition to saving energy, our protocol is also designed to recover from occasional packet loss and slot collisions (due to physical layer effects such as fading). The protocol is designed to be simple and robust enough to be implemented on current sensors (e.g., [3]).

In Section II, we present the target sensor network for our protocols. Related work is discussed in Section III. Section IV describes our proposed protocols. Section V discusses issues with the MAC protocol and upper layers in the network stack. Future work is presented in Section VI. We conclude the paper with Section VII.

II. Sensor Network Overview

We now describe the type of sensor network for which our protocol is targeted. While the described network may not be general enough for some objectives, we feel it would be useful in a large number of applications (e.g., the monitoring applications mentioned in [7]).

There are M data sinks, where $M \geq 1$, and N sensor nodes, where $N \gg M$. The sinks have an out-of-band means of aggregating their received data, so the sensors only need to anycast their data to a sink. The sinks are not energy constrained like the sensors. All data packets in the system have the same size and take T_{data} time to transmit over the channel[1]. All ACK packets take T_{ctrl} time to transmit. Based on these values, time is divided into slots of length T_{slot}, where T_{slot} is slightly larger than $(T_{data} + T_{ctrl})$ for reasons discussed in Section IV.

Let $T_{cycle} = K_{cycle} \times T_{slot}$, where K_{cycle} is an integer value that is specified when the network is deployed. We assume that packets are sent from a sensor at a deterministic, periodic rate. The rate may change over time, but changes are assumed to be infrequent. For an example of such behavior, consider sensors that periodically report their samples at a low rate in steady-state. When an event occurs, the detecting sensors will temporarily increase their sending rate to provide more detailed information about the event.

If a sensor chooses to send information to the sinks, it must send at a rate between R_{min} and R_{max}. Sensors may choose to send at a rate less than R_{min}, but it will not result in extra energy savings. The minimum rate is once per cycle, $R_{min} = \frac{1}{T_{cycle}}$. The maximum rate can be set when the network is deployed, but at most one data packet can be sent per slot, so $R_{max} \leq \frac{1}{T_{slot}}$.

The sensors are assumed to be periodically synchronized by the sinks such that the difference in clocks between any two nodes in the network is always less than Δ. To do this, we assume the sinks have enough energy to periodically broadcast with sufficient power to reach all sensors and provide a synchronization epoch. Thus, the broadcast must be done frequently enough to maintain a desired level of synchronization.

[1]If larger packets are needed, fragmentation can be done at the application layer.

Based on hardware specifications, it is possible to maintain a synchronization level based on a clock drift of around 10 to 40 parts-per-million (ppm) [8], [9]. Thus, to maintain $\Delta = 1$ ms, beacons would need to be sent every 25 to 100 seconds. In practice, even better results have been observed: sensors only need to be synchronized once every 13 minutes for $\Delta = 1$ ms (i.e., about 1.3 ppm drift) [10].

Currently, sensor hardware [3] can achieve a data rate of about 40 kbps. A widely used sensor operating system, TinyOS [11], uses 30 byte data packets, with an additional 34 bytes for MAC and physical layer headers. Thus, to transmit a data packet and its headers takes about 13 ms on this platform. If $\Delta \leq 1$ ms is maintained, then $\Delta \leq \frac{1}{13}T_{data}$ for current sensor hardware and software.

III. Related Work

IEEE 802.11 [6] specifies a simple Power Save Mode (PSM). Nodes are assumed to be time-synchronized and awake at the beginning of each *beacon interval*. After waking up, each node stays on for a period of time known as the *ATIM window*. During the ATIM window, since all nodes are guaranteed to be on, packets are advertised that have arrived since the previous beacon interval (or could not be sent in the previous beacon interval). These advertisements take the form of ATIM packets. More formally, when a node has a packet to advertise, it sends an ATIM packet to the intended destination during the ATIM window, following the rules of IEEE 802.11's CSMA/CA mechanism. In response to receiving an ATIM packet, the destination will respond with an ATIM-ACK packet (unless the ATIM specified a broadcast or multicast destination address). When this ATIM handshake has occurred, both nodes will remain on after the ATIM window and attempt to send their advertised data packets before the next beacon interval, subject to CSMA/CA rules. If a node remains on after the ATIM window, it must keep its radio on until the next beacon interval. If a node does not receive an ATIM or ATIM-ACK (assuming unicast advertisements), it will enter sleep mode at the end of the ATIM window until the next beacon interval. This process is illustrated in Figure 1. The dotted arrows indicate a "causes" relationship. Node **A** sends a data packet to **B** while **C**, not receiving any ATIM packets, returns to sleep for the rest of the beacon interval.

While 802.11 PSM is a simple protocol to implement (if synchronization can be achieved) there are some disadvantages. First, the ATIM window and beacon interval times must be adjusted for different traffic patterns [12], [13]. Also, nodes must remain on for the entire beacon interval if they have only one packet to send and/or receive. This can be wasteful in terms of energy. For example, if packet interarrival time is less that one beacon interval, then the node will never sleep,

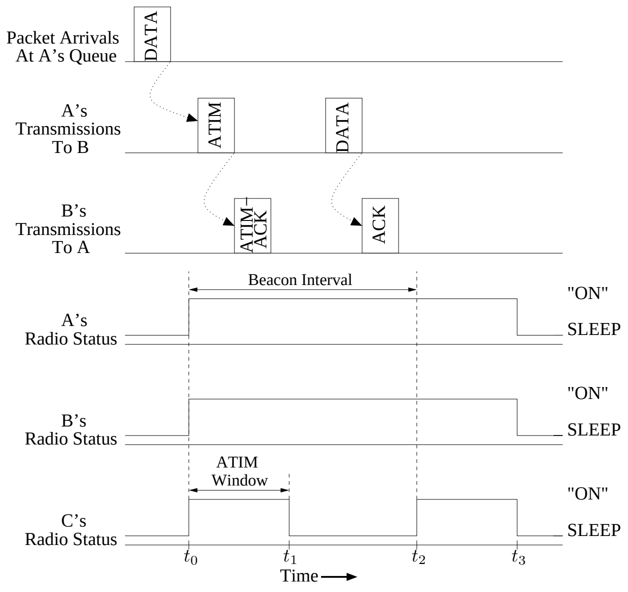


Fig. 1. IEEE 802.11 power save mechanism [6].

regardless of how large the beacon interval is compared to the time to transmit one data packet. Finally, the protocol uses 802.11's CSMA/CA mechanism to avoid collisions. This may not work well when a large number of competing nodes try to send their packets immediately after the ATIM window rather than spreading the packet transmissions out uniformly over the beacon interval.

Research in power save protocols at the MAC layer has taken three general directions. First, there is a scheduling approach whereby packet transmission times are carefully chosen to avoid collisions. This approach lends itself well to power save since nodes can sleep when they are not scheduled to communicate. Second, there are protocols which attempt to allow all but a small subset of nodes to sleep. The members of this subset are periodically rotated and chosen to preserve multihop paths in the network. Finally, work has been done with "wakeup" channels where nodes have an out-of-band means of waking their sleeping neighbors. We will mention work from each category, but focus on the first category since it is most related to our protocol.

The PAMAS protocol [14] was among the first power save protocols. It adapts basic mechanisms of IEEE 802.11 [6] to a two-radio architecture. Nodes can sleep when other nodes are transmitting or their transmission would interfere with the transmissions of neighbors. However, PAMAS ignores the idle listening problem. S-MAC [2] adopts the PAMAS protocol to single-radio nodes and adds a simple scheduling mechanism to reduce idle listening. Nodes attempt to synchronize sleep schedules with surrounding neighbors by periodically broadcasting their schedule. T-MAC [15] extends S-MAC by adjusting the length of time sensors are awake between sleep intervals based on communication of nearby neighbors. In [16], a large number of non-interfering frequency bands are used for a primarily FDMA scheduling approach.

The TRAMA protocol [17] is similar to our protocol because it uses collision-free packet scheduling for energy efficiency. Nodes periodically awake to exchange broadcasts and learn their two-hop neighborhood. Based on this knowledge, nodes periodically reserve future slots for backlogged traffic. A hash-based priority scheme is then used so that only one node in a two-hop neighborhood will transmit in a given slot. Our protocol is different than TRAMA because it attempts to schedule long-lived, end-to-end, periodic flows instead of scheduling recently received packets on a hop-by-hop basis. Also, our protocol does not need to maintain consistent two-hop neighborhood information and results in a much simple scheduling algorithm.

In [18], a TDMA scheduling protocol is presented for constant bitrate (CBR) traffic. In this protocol, nodes listen for data and ACK packets from neighboring nodes. Whenever one of these packets is overheard in a slot, the node marks that slot as "used." Thus, when a node wants to schedule its own flow, it can look at its slot table and find an open slot to use. However, this approach is not energy efficient because nodes must constantly listen to the channel to keep their slot table up-to-date.

Another related protocol is presented in [19]. Collision-free, deterministic, periodic flows are scheduled toward a data sink. Nodes maintain a table of times when they should wakeup to send and receive. When a node wishes to schedule a new flow on a link, nodes awake at a specified time and the sender will try to send an *RSETUP* packet if no data transmission is sensed in the slot. If the RSETUP packet is received successfully, an ACK is sent back and the slot is scheduled. However, if the RSETUP causes a collision at the receiver, no ACK is sent. In this case, the sender must try to send the RSETUP packet in a different slot because using the current slot would interfere with an existing flow.

One disadvantage of this approach is broadcasts/multicasts cannot be scheduled since explicit ACKs are required to create a schedule. This is particularly detrimental because nearly all ad-hoc routing and neighbor discovery protocols rely on broadcasts. Our protocol is able to support broadcast scheduling.

Additionally, this protocol may cause significant data packet loss while trying to setup a flow by causing multiple collisions with the RSETUP packet. Also, this protocol may destroy an existing flow while setting up a new flow. This situation is demonstrated by the topology in Figure 2. Assume **B** is sending a previously scheduled flow to **A** through **C**. At some point, **D** attempts to schedule a flow to **A** through **E**. If **D** sends its RSETUP packet at the same time **B** sends its scheduled DATA packet, a collision will occur at **C** (**D** may have been sleeping when **B** and **C** originally scheduled their flow). However, **E** is still able to respond with an ACK to **D**. Thus, the flow **D**→**E** will be scheduled in the same slot as **B**→**C**, causing deterministic packet loss at **C**. This problem and the potential multiple collisions caused by an RSETUP do not exist in our protocol.

The approach of keeping a small subset of nodes awake for

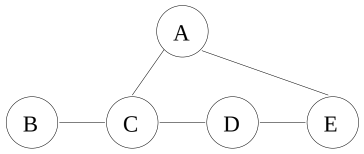


Fig. 2. Example topology to demonstrate flow scheduling problem in [19].

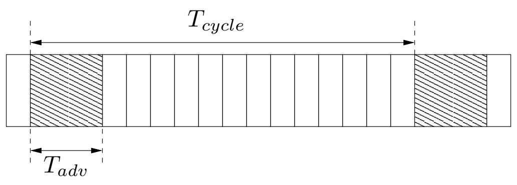


Fig. 3. Timing Diagram for T_{cycle}.

multihop communication is taken in SPAN [20], GAF [21], and ASCENT [22]. Each of these protocols is designed for dense network where a large fraction of nodes can sleep without significantly reducing connectivity. In [23], the subset of nodes discovered via an on-demand routing protocol remain awake to communicate at a low latency while allowing all other nodes to sleep.

The PicoRadio [24]–[26] design uses a special hardware to do paging on a low-power wakeup channel. STEM [27], [28] uses two radios, to periodically listen on a paging channel while allowing both radios to sleep a large fraction of the time. In [29], [30], STEM is combined with a scheduling technique to further minimize energy consumption. In [31], a paging protocol for use with centralized access points is implemented from off-the-shelf hardware.

Though not related to power save protocols, we propose the use of busy tones in our protocol. This technique has been proposed previously to avoid interference with on-going communication when nodes are mobile [32], [33]. Our use of busy tones is different because it is used to avoid collisions in packet scheduling.

IV. Protocol Description and Discussion

Our goal is to schedule sensor-to-sink flows by scheduling a slot for transmission and reception at each hop along the path that does not interfere with existing flows. To achieve this goal, we propose two similar protocols called On-Demand Scheduling (ODS) and Busy Tone On-Demand Scheduling (BTODS). Both are TDMA scheduling protocols and do not require nodes to maintain consistent two-hop neighborhood information, a source of control overhead. Additionally, the protocols allow broadcasts and multicasts to be scheduled for use in upper layer protocols.

We begin by describing the common part of both protocols. ODS and BTODS differ in how nodes behave within data slots. In Section IV-A we describe the BTODS protocol for data slots. Section IV-B discusses how flow collisions are detected and handled. Section IV-C describes the ODS protocol. The comparative advantages and disadvantages of ODS and BTODS are discussed in Section IV-D.

A “flow” refers to a MAC layer flow in which data packets are sent every T_{cycle} time. If an upper layer needs to send at a rate faster than once every T_{cycle}, multiple MAC layer flows can be scheduled. If data needs to be send at a rate that is not a multiple of T_{cycle}, then extra slots can be reserved in each cycle. For example, if three data packets need to be sent every two cycles, then two slots per cycle can be reserved. This allows up to four data packets to be sent every two cycles. Thus, one of the reserved slots could be used to transmit a “dummy” packet (and keep the slot reserved for future cycles) while the other three slots are used to send data. We assume a routing layer exists. In Section V-A, we describe a simple routing protocol that can be used with our protocols.

After each data packet, an ACK is sent if the data is received correctly. If a data packet is not received in a scheduled slot, a NACK packet is sent. The NACK packet has a bit which indicates whether a collision was occurred or the packet was just received with an error or lost. The sender can then respond appropriately. Reliability issues are discussed further in Section V-C.

Each sensor maintains a table of K_{cycle} entries (recall that $T_{cycle} = K_{cycle} \times T_{slot}$) indicating which slots it should be awake to transmit and which it should be awake to receive data from a neighbor. When a data packet is sent, there is a *ONE-SHOT* bit piggybacked on the header. If this bit it set, it indicates this data packet is not part of a flow and, hence, it will not be included in the sender or receiver's schedule table.

At the beginning of every cycle all nodes wake up for a few slots, called the *FLOW-ADV window*, which takes a total time of T_{adv}. Thus, if a node has no packets to send or receive, it will idly listen for T_{adv}/T_{cycle} fraction of the time and sleep for the remaining $(T_{cycle} - T_{adv})/T_{cycle}$ fraction of the time. Presumably, $T_{adv} \ll (T_{cycle} - T_{adv})$. The timing diagram for this is shown in Figure 3.

When a node has a new flow to schedule, it must advertise it during the FLOW-ADV window by sending a *FLOW-ADV* packet. This is similar to 802.11's ATIM window except flows, instead of packets, are being advertised. Because there are only a few slots available during the FLOW-ADV window, it is assumed that few FLOW-ADV packets are sent each interval. To address contention, nodes can do random access during the FLOW-ADV window or choose the slot to send their FLOW-ADV packet uniformly at random among the slots available during the FLOW-ADV window. These advertisements require a *FLOW-ACK* similar to in 802.11's ATIM protocol (unless the FLOW-ADV is for a broadcast). The FLOW-ADV indicates how many flows the sender desires to schedule with the receiver in the upcoming interval. No data packets are sent during the FLOW-ADV window, so the window does induce some control overhead.

At the end of the FLOW-ADV window, all nodes that did not send or receive a FLOW-ADV return to sleep and follow their schedule table for waking up during the current interval. Nodes that received a FLOW-ADV will remain on until a slot has been scheduled for the number of flows specified in the

FLOW-ADV. If the nodes remain on for the entire T_{cycle} and are unable to schedule a flow, this results in a *route error* (RERR) and the flow must find a different path to be scheduled on. For now, we will assume all flows are able to be scheduled. Handling RERRs is left up to the routing layer.

Flows can be destroyed explicitly or implicitly. A node can transmit a *FLOW-DEL* message during the flow's scheduled data slot (or set a FLOW-DEL bit in the data header of the flow's last data packet). The FLOW-DEL message will traverse along the path in this manner and each node will remove the flow from its schedule table. Additionally, each node that does not receive a scheduled transmission in $K_{RX\_flow-del}$ consecutive intervals will delete the flow from its schedule table[2]. If a sender does not receive an ACK for a scheduled flow in $K_{TX\_flow-del}$ consecutive intervals, it will delete the current scheduled sending slot from its table and attempt to find a new path for the flow at the routing layer.

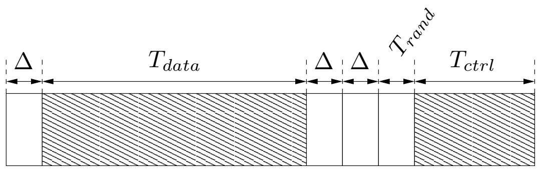
A. BTODS

Using busy tones requires that the channel can be divided into two sub-channels, one for data and one control channel for busy tones. Busy tones do not need to be demodulated, but neighbors must be able to detect the presence of the busy tone signal on the control channel. It is assumed that the transmission ranges of control channel and data channel are chosen to keep interference sufficiently low. Thus, busy tones are used to avoid the hidden terminal because receivers can emit a busy tone while they are receiving data to let all its neighbors know that a transmission would interfere with the receiver's packet reception. An alternative to using a busy tone would be to equip sensors with two identical radios operating at different frequencies (e.g., [27]). One radio could transmit data, while the control radio transmits dummy packets periodically during packet reception. Surrounding nodes could detect on-going receptions by listening long enough on their control radio to hear a dummy packet or else decide no neighbors are receiving. For the rest of this paper, we will assume busy tones are used.

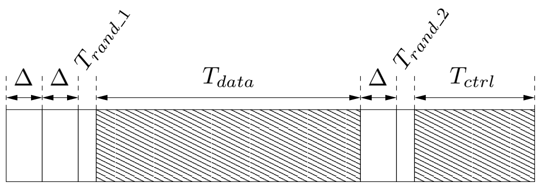
In BTODS, a node will transmit a busy tone under the following conditions:

- It is receiving on the data channel.
- It overhears another node sending on the data channel.

We now discuss how BTODS finds non-interfering slots. First, we will describe the structure of slots in BTODS. Figure 4 shows the timing diagrams for BTODS slots with unicast data. In this figure, T_{rand_1} is chosen uniformly at random from $(T_{min}, T_{rand}]$ and $T_{rand\_1} + T_{rand\_2} = T_{rand}$ to keep the slots aligned[3]. T_{min} is chosen to be long enough to allow a node's busy tone to be propagated to its two-hop neighbors. For example, if node **B** is a one-hop neighbor of node **A**, then T_{min} must be long enough that **B** can detect **A**'s busy tone, transmit a busy tone, and have it detected by all of **B**'s one-hop neighbors. For broadcasts and multicasts, the timing diagram is the same except that no ACKs are sent during T_{ctrl}.



(a) Previously scheduled data.



(b) Attempting to schedule data.

Fig. 4. Timing diagram for slots in BTODS.

Every slot begins with an idle listening period of Δ to account for synchronization errors. As we can see from Figure 4(a) and Figure 4(b), data that has been previously scheduled in a slot will receive priority over data which is attempting to be scheduled in the slot. This is because previously scheduled data is sent immediately after the Δ listening period whereas the data attempting to be scheduled will wait $(2\Delta + T_{rand\_1})$ before attempting to sent its data. The length of T_{rand_1} is chosen such that a node has enough time to carrier sense the control channel as idle before sending. If any signal is detected, the nodes will try to find a different slot to communicate.

From Figure 4, we make three claims about the correctness of the protocol:

- *A previously scheduled flow will always send its data before a new flow*: From Figure 4(b), the earliest a new flow will send its data is $(2\Delta + T_{rand\_1})$ after the beginning of the slot. From Figure 4(a), the latest an existing flow will send its data (assuming its clock is Δ later than the clock of the node scheduling the new flow) is 2Δ. Thus, the previously scheduled flow will always send its data first.
- *An existing flow's data will never overlap with a new flow's ACK*: This assures that an ACK being sent by the receiver of an existing flow will not interfere with the data receiver of a new flow. From Figure 4(a), we see that the earliest an ACK for an existing flow can be sent is $(3\Delta + T_{rand} + T_{data})$. Assuming a Δ synchronization error, from Figure 4(b), the latest a new flow's data can end is $(3\Delta + T_{rand} + T_{data})$. Thus, the new flow will always finish sending its data before the existing flow begins sending its ACK.

[2]Alternatively, instead of using the criteria of missing packets for k consecutive intervals, we could use criteria of missing packets for any k of most recent n intervals.

[3]Thus, whatever value a node selects for T_{rand_1}, it will set $T_{rand\_2} = T_{rand} - T_{rand\_1}$. This ensures that the entire slot time is $(3\Delta + T_{rand} + T_{data} + T_{ctrl})$.

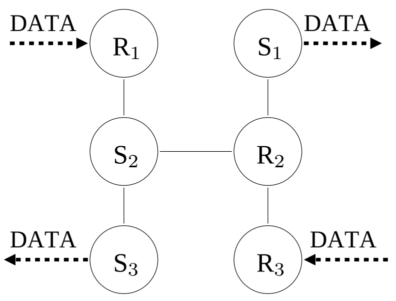


Fig. 5. Example topology for BTODS scheduling.

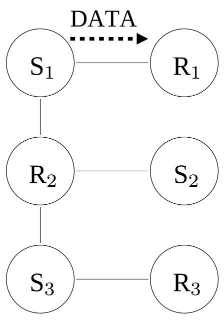


Fig. 6. Example topology for overly conservative scheduling.

- *An existing flow's data will never overlap with another existing flow's ACK*: This is similar to the previous claim. From Figure 4(a), we see that the earliest an ACK for an existing flow can be sent is $(3\Delta + T_{rand} + T_{data})$. Also from Figure 4(a), the latest the data can finish being sent is $(2\Delta + T_{rand} + T_{data})$, assuming a Δ synchronization error. To see why this property is important, consider nodes $\mathbf{S_2}$ and $\mathbf{S_3}$ in Figure 5. These nodes will be able to send data to their receiver and receive the corresponding ACK in the same slot. In particular, $\mathbf{S_3}$'s data transmission cannot interfere with $\mathbf{S_2}$ receiving an ACK from $\mathbf{R_2}$ in the same slot (and vice versa).

We use the topology in Figure 5 as an example of how BTODS works. Assume that node $\mathbf{S_2}$ wants to schedule a flow to send data to $\mathbf{R_2}$. Nodes $\mathbf{S_1}$ and $\mathbf{S_3}$ already have previously scheduled flows in which they are transmitting. Nodes $\mathbf{R_1}$ and $\mathbf{R_3}$ already have previously scheduled flows in which they are receiving. After $\mathbf{S_2}$ and $\mathbf{R_2}$ have completed their FLOW-ADV/FLOW-ACK exchange, both will remain on until they can find an open slot in which $\mathbf{S_2}$ can send a packet and receive an ACK from $\mathbf{R_2}$ without interfering with existing flows. BTODS requires the sender, $\mathbf{S_2}$, to refrain from attempting to schedule and send a data packet in any slot in which it detects a busy signal. Obviously, the sender will also refrain from transmitting in slots in which it will receive a previously scheduled flow.

Now, using Figure 5, we will explain why the slot BTODS schedules for the $\mathbf{S_2} \rightarrow \mathbf{R_2}$ communication will not interfere with existing communication in the slot. In the slot in which $\mathbf{R_1}$ is receiving, the flow will not be scheduled because $\mathbf{S_2}$ will refrain from sending data when it hears $\mathbf{R_1}$'s busy tone. In the slot in which $\mathbf{S_1}$ is sending data, the new flow will not be scheduled because $\mathbf{R_2}$ will overhear $\mathbf{S_1}$'s transmission in the slot and emit a busy tone. $\mathbf{S_2}$ will detect this busy tone and refrain from sending data in the slot. The communication between $\mathbf{S_2} \rightarrow \mathbf{R_2}$ can be scheduled in the same slot as $\mathbf{S_3}$'s transmission or $\mathbf{R_3}$'s reception without interfering. As discussed previously, the slot structure in Figure 4 ensures that the beginning of $\mathbf{R_3}$'s ACK transmission will not interfere with the end of $\mathbf{R_2}$'s data reception (and vice versa). The slot structure also ensures that the end of $\mathbf{S_3}$'s data transmission will not collide with the beginning of $\mathbf{S_2}$'s ACK reception (and vice versa).

We can see that BTODS can easily support broadcast/multicast scheduling as well. When a node sends a FLOW-ADV for a broadcast packet, all of its neighbors will remain on after the FLOW-ADV window. The sender will only transmit its packet in a slot in which none of its neighbors are transmitting busy tones or data. Note that because the intended receivers will transmit a busy done when they overhear or receive a previously scheduled data packet, the absence of a busy tone *and* absence of a data transmission implies that the sender's broadcast will be received by all its neighbors in the chosen slot. For unicast data, if a sender attempts to schedule a flow in a slot where the intended receiver is transmitting to a different node, the intended receiver will not reply with an ACK and the sender will try a different slot.

One problem with BTODS is it may be too conservative it its scheduling. This means non-interfering flows may not always be scheduled in the same slot when possible. Consider the topology in Figure 6.

Assume that the existing flow $\mathbf{S_1} \rightarrow \mathbf{R_1}$ is using a slot and $\mathbf{S_2}$ and $\mathbf{S_3}$ are trying to schedule a flow to $\mathbf{R_2}$ and $\mathbf{R_3}$, respectively, in the same slot. Obviously, it would be possible for the flows $\mathbf{S_1} \rightarrow \mathbf{R_1}$ and $\mathbf{S_3} \rightarrow \mathbf{R_3}$ to use the same slot without interfering. However, in the given slot, $\mathbf{R_2}$ will overhear $\mathbf{S_1}$'s data transmission and, in response, transmit a busy tone to let $\mathbf{S_2}$ know it cannot send data in the current slot. However, $\mathbf{S_3}$ will also detect $\mathbf{R_2}$'s busy tone and falsely assume it cannot send to $\mathbf{R_3}$ during the current slot.

We expect that flow scheduling will be an infrequent task and, therefore, the situation in Figure 6 will rarely ever occur. To avoid deterministic behavior, if a node cannot schedule its flow over an entire T_{cycle} time, it could choose a random number of cycles to wait before trying again. In this case, $\mathbf{S_2}$ and $\mathbf{S_3}$ in Figure 6 should eventually choose a cycle in which the other is not trying to schedule its data. Also, once $\mathbf{S_2}$ finds a slot to schedule its data, $\mathbf{S_3}$ will be able to schedule its flow in the same slot as $\mathbf{S_1}$.

B. Handling Flow Collisions

Interfering flows could still be scheduled in the same slot in the following situations:

1) The senders of two flows attempting to be scheduled choose T_{rand_1} such that their data transmissions start at virtually the same time.
2) Physical layer effects (e.g., fading on the busy tone channel) allow nodes to determine that the channel is "free" when, in reality, the slot is already being used by an interfering flow.
3) The topology changes (e.g., an obstacle is removed) to create an interfering link that was not there when flows were originally scheduled.

The common characteristic of all these situations is that we assume they occur infrequently and nondeterministically. However, our protocol still needs to specify how to detect interfering flows and recover.

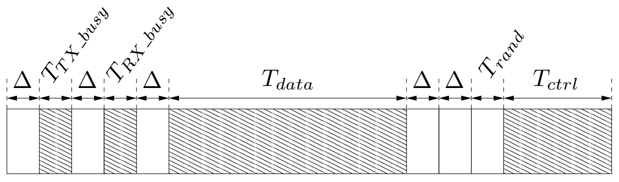
Consider Situation 1 using Figure 5. Assume that $\mathbf{S_{R_1}}$ (not shown) is attempting to schedule a flow with $\mathbf{R_1}$ and $\mathbf{S_2}$ is attempting to schedule a flow with $\mathbf{R_2}$ in the same slot. In this situation, a collision will occur at $\mathbf{R_1}$. Thus, $\mathbf{R_1}$ will not respond with an ACK (and hence not schedule the flow). When $\mathbf{S_{R_1}}$ does not receive an ACK, it will try to schedule the flow in a different slot. If $\mathbf{S_1}$ does receive its ACK, it will schedule its flow in the current slot. Otherwise, $\mathbf{S_1}$ will also attempt a different slot.

Situation 2 and Situation 3 can be handled similarly. In Figure 5, assume that $\mathbf{S_1}$ is sending to $\mathbf{R_{S_1}}$ (not shown) in the same slot as $\mathbf{S_2}$ is sending to $\mathbf{R_2}$ before the link from $\mathbf{S_1} \leftrightarrow \mathbf{R_2}$ exists. In this case, when the interference exists, $\mathbf{R_2}$ will suffer a collision during T_{data} and respond with a NACK indicating a collision to $\mathbf{S_2}$. Upon receiving this NACK, $\mathbf{S_2}$ will attempt to schedule the flow in a different slot. Also, if a collision occurs during T_{ctrl}, the sender will attempt to reschedule the flow. In this manner, our protocols can recover from occasional flow collisions.

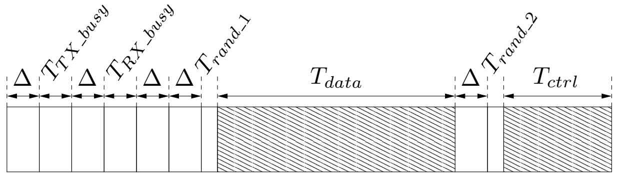
C. ODS

As we will discuss in Section IV-D, there are some disadvantages to using BTODS. Therefore, in this section, we present ODS which is similar to BTODS, but does not use busy tones. The slot structure of ODS is shown in Figure 7.

As explained in Section IV-A, T_{rand_1} is chosen uniformly at random from $(0, T_{rand}]$ and $T_{rand\_1} + T_{rand\_2} = T_{rand}$ to keep the slots aligned. Busy tones are not used in ODS, so extra periods are added for nodes to indicate they will be busy sending or receiving in the current slot. These slots take T_{TX_busy} and T_{RX_busy}, respectively, as shown in Figure 7. When a node is scheduled to send data in the current slot, it will transmit on the data channel during the TX_busy period. A node will transmit on the data channel during the RX_busy period if either it is scheduled to receive data in the current slot or it heard another node transmit during the TX_busy period. The latter case indicates another node is already scheduled to transmit in the current slot. Thus, a node which detects the data channel busy during the TX_busy period must indicate that it will not be able to receive in the current slot to all potential senders, which it does during the RX_busy period.



(a) Previously scheduled data.



(b) Attempting to schedule data.

Fig. 7. Timing diagram for slots in ODS.

ODS can handle flow collisions using the methods discussed in Section IV-B.

Based on this description, we make the following five claims about ODS:

- *The TX_busy period will never overlap with another node's RX_busy period*: From Figure 7(a), the earliest a node's RX_busy period will begin is $(2\Delta + T_{TX\_busy})$. The latest another node's TX_busy period can end, assuming a drift of Δ, is $(2\Delta + T_{TX\_busy})$. Thus, nodes will always be able to tell whether the data channel is busy during the TX_busy period or RX_busy period without overlap between the two.
- *The RX_busy period will never overlap with another node's data communication*: From Figure 7(a), the earliest a data communication can begin is $(3\Delta + T_{TX\_busy} + T_{RX\_busy})$. Assuming a Δ drift, the latest a node's RX_busy period can end is $(3\Delta + T_{TX\_busy} + T_{RX\_busy})$. Thus, a node's data reception will never suffer interference due to transmissions in the RX_busy period.
- *A previously scheduled flow will always send its data before a new flow*: From Figure 7(b), the earliest a new flow will send its data is $(4\Delta + T_{TX\_busy} + T_{RX\_busy} + T_{rand\_1})$ after the beginning of the slot. From Figure 7(a), the latest an existing flow will send its data (assuming its clock is Δ later than the clock of the node scheduling the new flow) is $(4\Delta + T_{TX\_busy} + T_{RX\_busy})$. Thus, the previously scheduled flow will always send its data first.
- *An existing flow's data will never overlap with a new flow's ACK*: This assures that an ACK being sent by the receiver of an existing flow will not interfere with the data receiver of a new flow. From Figure 7(a), we see that the earliest an ACK for an existing flow can be sent is $(5\Delta + T_{TX\_busy} + T_{RX\_busy} + T_{rand} + T_{data})$. Assuming a Δ synchronization error, from Figure 7(b), the latest a new flow's data can end is $(5\Delta + T_{TX\_busy} + T_{RX\_busy} + T_{rand} + T_{data})$. Thus, the new flow will always finish sending its data before the existing flow begins sending its ACK.

- *An existing flow's data will never overlap with another existing flow's ACK*: This is similar to the previous claim. From Figure 7(a), we see that the earliest an ACK for an existing flow can be sent is $(5\Delta + T_{TX\_busy} + T_{RX\_busy} + T_{rand} + T_{data})$. Also from Figure 7(a), the latest the data can finish being sent is $(4\Delta + T_{TX\_busy} + T_{RX\_busy} + T_{rand} + T_{data})$, assuming Δ synchronization error.

ODS works similar to BTODS on the topology in Figure 5, assuming $\mathbf{S_2}$ wants to schedule a flow with $\mathbf{R_2}$. Nodes $\mathbf{S_2}$ and $\mathbf{S_3}$ can transmit in the same slot without interference. Similarly, $\mathbf{R_2}$ and $\mathbf{R_3}$ can receive in the same slot without interference. $\mathbf{S_2}$ will decide not to use a slot if it detects the data channel as busy during the RX_busy period. If $\mathbf{S_2}$ detects a collision during the RX_busy period, the data channel is considered busy. $\mathbf{S_2}$ can still use a slot if it detects the TX_busy period busy but not the RX_busy period.

In a slot in which $\mathbf{R_1}$ is receiving, $\mathbf{S_2}$ will detect $\mathbf{R_1}$'s transmission on the data channel during the RX_busy period and will not use the slot. In a slot in which $\mathbf{S_1}$ is transmitting, $\mathbf{R_2}$ will detect $\mathbf{S_1}$'s transmission during the TX_busy period. In response, $\mathbf{R_2}$ will transmit during the RX_busy period. $\mathbf{S_2}$ will detect $\mathbf{R_2}$'s transmission during the RX_busy period and defer from using the current slot.

ODS suffers from the same problem as BTODS of being overly conservative in scheduling flows. For example, consider the situation described in Section IV-A using Figure 6. In the given slot, $\mathbf{R_2}$ will overhear $\mathbf{S_1}$'s transmission during the TX_busy period. In response, $\mathbf{R_2}$ will transmit during the RX_busy period to let $\mathbf{S_2}$ know it cannot send data in the current slot. However, $\mathbf{S_3}$ will also detect $\mathbf{R_2}$'s data transmission during the RX_busy period and falsely assume it cannot send to $\mathbf{R_3}$ during the current slot.

D. Comparison of ODS and BTODS

There are advantages and disadvantages to both ODS and BTODS. As discussed in Section IV-A and Section IV-C, both protocols may be too conservative in scheduling flows. It is our assumption that flow scheduling will be relatively infrequent and, therefore, being conservative will not greatly effect protocol performance.

The main disadvantage of BTODS is it requires the hardware capability to provide two non-interfering channels, either by splitting a channel or using two separate radios. If the channel is split (e.g., using a small portion of the bandwidth for the busy tone), then the bitrate of the data channel may be slightly reduced and the busy tone may be more susceptible to the effects of fading [34]. Also, the radio must be able to receive on the data channel while simultaneously transmitting on the busy tone channel, which may be difficult to achieve. If two radios are used, then twice as much bandwidth is needed and energy will be consumed by both radios. Also, by using a separate busy tone channel, the channel conditions on the data and busy tone channel may differ. BTODS requires that the busy tone and data channel have the same receive and carrier sense range, which may be difficult to achieve in practice. ODS does not suffer these problems because it uses one channel.

The main disadvantage of ODS is the longer slot time required by the protocol relative to BTODS. As shown in Figure 4 and Figure 7, one ODS slot takes $(2\Delta + T_{TX\_busy} + T_{RX\_busy})$ longer than a BTODS slot. Thus, less time is devoted to data transmission in ODS. There is more control overhead in terms of time and transmissions because nodes must transmit during the TX_busy and RX_busy periods when appropriate.

V. Interactions with Upper Layers

A. Routing

For routing, we are focused on sensor-to-sink communications. Thus, we do not need an approach as general as DSR [35] or AODV [36]. Instead, a simple approach for our scenario is to periodically broadcast beacons from each of the M sinks. We assume that the TTL of the broadcast is large enough to reach all the nodes in the network and that beacon sequence numbers are maintained such that each node sends only one copy of each broadcast and suppresses all others. This type of routing, which has been proposed previously [37], [38], works well with BTODS and ODS because sinks can schedule the beacons to propagate through the network at periodic intervals. The disadvantage of this approach is when the topology changes, nodes must wait a beacon interval to learn of the change (as opposed to DSR or AODV, which adapt faster). However, we feel that this beacon-based approach is appropriate for sensor networks since the topology is fairly static and the sensors can route to any of the M sinks. Therefore, if a path to one of the sinks becomes unusable, a sensor can route to a different sink until a new beacon is heard.

The beacons do not have to be sent every T_{cycle} seconds. Instead, they can be sent every $(K_{beacon} \times T_{cycle})$ time units, where K_{beacon} in an integer value. If K_{beacon} is small, it may be worthwhile to actually schedule a slot for the beacons and defend the slot with dummy packets (if $K_{beacon} \neq 1$) as discussed in Section IV. However, if K_{beacon} is large, nodes could just send broadcasts with the ONE-SHOT bit set.

If a RERR needs to be propagated back to the flow's source, this can be done in BTODS and ODS by piggybacking the RERR information on NACK packets sent during the T_{ctrl} period of a slot.

B. Congestion Control

Because each node maintains a schedule table of slots which it is transmitting and receiving, this gives us an opportunity to do congestion control proactively based on how full a node's schedule table is. Proactive congestion control is preferable to the more traditional approach of rapidly decreasing sending when packet loss is detected because it does not induce packet loss or assume all loss is due to congestion. Each node can calculate f_{full}, the fraction of slots in its schedule table which are being used (i.e., $f_{full} \in [0, 1]$). Congestion notification information can be piggybacked on the ACKs sent during T_{ctrl}.

One disadvantage of this simple congestion control scheme is that a large f_{full} value is a sufficient, but not necessary, condition for indicating the node will have a smaller probability of scheduling new flows. If f_{full} is small, a node may still have a small probability of scheduling new flows if its neighbors' scheduled flows would interfere with most of the node's open slots in its schedule table.

C. Reliability

The BTODS and ODS protocols provide notification when a packet is not received by the ACK/NACK sent during T_{ctrl}. However, BTODS and ODS do not retransmit the packet at the MAC layer like IEEE 802.11. If retransmissions at the MAC layer are desired, this could be done in a couple of ways. First, there could be time set aside in each cycle which is reserved for retransmissions. When a packet is not received in an assigned slot, the nodes could turn on during this reserved time and compete for retransmission (similar to competition during the FLOW-ADV window). However, this incurs extra overhead and requires tuning based on the number of expected retransmissions per cycle.

An alternative is to have the nodes remain on when the packet is not received in an assigned slot and attempt to retransmit the packet in another slot with the ONE-SHOT bit set. This allows the retransmission to be done in an on-demand fashion. However, there is no guarantee that an open slot will be found and the nodes may have to remain on for many slots to listen to find an open slot.

Obviously, finding and using high-quality links is helpful in increasing reliability. This is beyond the scope of the MAC protocol. However, if links are thought to be of poor quality, BTODS and ODS could attempt to compensate by scheduling multiple slots for the flow rather than just one. This creates a tradeoff in reliability and energy usage.

VI. Future Work

In the future, we plan to implement the protocols presented in this paper in sensors [3]. We believe that the protocols should be reasonably effective in scheduling flows at the MAC layer to conserve energy and reduce interference. However, there are many future avenues of work that are possible to improve BTODS and ODS.

One possible improvement to BTODS and ODS is to investigate how T_{adv} can dynamically change based on the amount of flows that need scheduled. For example, one can envision a scenario when flows are rarely scheduled except in response to an event. When the event occurs, many nodes will attempt to schedule flows. Thus, we would like T_{adv} to be small in a steady-state, but increase quickly when events are detected. A challenge is keeping all of the nodes synchronized in this situation. Similar work has been done with the ATIM window in IEEE 802.11 [12], [13].

Another area for future work is determining how to best fragment the schedule table. For example, nodes may favor scheduling flows in consecutive slots to reduce the energy cost of transitioning on and off multiple times. However, because of previous, existing flows, the sender and receiver pair may not always be able to choose a slot which minimizes schedule table fragmentation for each node.

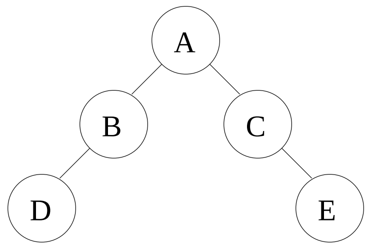


Fig. 8. Example topology to demonstrate BTODS scheduling effect on latency.

Broadcast scheduling is another direction to improve BTODS and ODS. If a broadcast cannot be scheduled for a sender, it may be possible to send two multicast packets instead. We plan to investigate how broadcast messages could be scheduled as multiple multicast packets.

Finally, there is the issue of how scheduling affects buffering requirements at intermediate nodes. We assume that a sensor can schedule when it generates a data packet so that it does not have to buffer a packet long before sending it in its scheduled slot.

If a node receives a packet in slot i and sends it in slot $i+1$, the node buffers the packet less time than if it must wait until slot $i+j$ to send it (where $j>1$). This also has an effect on the sensor-to-sink latency. Consider the topology in Figure 8 where **D** and **E** both want to create a flow to **A**. If the **D**→**B** transmission is scheduled in slot i and the **E**→**C** transmission is scheduled in slot i, then the **B**→**A** transmission can be scheduled in slot $i+1$, but the **C**→**A** transmission must be scheduled in slot $i+2$. Thus, the **D**→**A** flow has latency of 2 slots whereas the **E**→**A** flow has a latency of 3 slots (where the packet remains buffered at **C** for 1 slot). If, however, the **E**→**C** transmission had been scheduled during slot $i+1$, both flows could have a 2 slot delay with no intermediate buffering (again, this assumes that **E** can adjust its data packet generation to create the packet in slot i instead of $i+1$).

In addition to these proposed directions of future work, there should be more extensive work on the interactions with upper layers discussed in Section V. The upper layer interactions should be specified in more detail and the tradeoffs more fully explored.

VII. Conclusions

We have presented an on-demand TDMA MAC protocol for sensor networks. We have described two variants of this protocol, ODS and BTODS. The major difference is one assumes that a separate, similar channel is available for busy tones. Both protocols are designed to schedule sensor-to-sink flows while reducing energy consumption due to collisions, overhearing, control overhead, and idle listening. Sensors schedule slots to send or receive data and then sleep for the remaining slots in each cycle. BTODS and ODS allow nodes to find slots which do not interfere with existing flows in their vicinity. We discussed how the protocols interact with upper layers in the network stack as well as areas of future work. We believe BTODS and ODS represent light-weight protocols that can be implemented in many sensor networks to allow efficient data transmission while reducing energy consumption.

REFERENCES

[1] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh, and D. Rubenstein, “Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet,” in *ACM Architectural Support for Programming Languages and Operating Systems (ASPLOS) 2002*, October 2002.

[2] W. Ye, J. Heidemann, and D. Estrin, “An Energy-Efficient MAC Protocol for Wireless Sensor Networks,” in *IEEE Infocom 2002*, June 2002.

[3] MICA2 Mote Datasheet, `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0042-01_A_MICA2.pdf`.

[4] S. Doshi and T. X. Brown, “Minimum Energy Routing Schemes for a Wireless Ad Hoc Network,” in *IEEE Infocom 2002*, June 2002.

[5] C.-Y. Wan, S. B. Eisenman, and A. T. Campbell, “CODA: Congestion Detection and Avoidance in Sensor Networks,” in *ACM SenSys 2003*, November 2003.

[6] IEEE 802.11, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999.

[7] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “The Design of an Acquisitional Query Processor For Sensor Networks,” in *ACM SIGMOD 2003*, June 2003.

[8] S. Ganeriwal, R. Kumar, and M. B. Srivastava, “Timing-sync Protocol for Sensor Networks,” in *ACM SenSys 2003*, November 2003.

[9] EM Microelectronic EM7602 Oscillator Datasheet, `http://www.emmarin.com/webfiles/product/other/em7602_fs.pdf`.

[10] John A. Stankovic, “A Network Virtual Machine for Real-Time Coordination Services: Network Architectures for Target Tracking,” `http://dtsn.darpa.mil/ixo/nest/day1/UVA_demo-presentation.ppt`.

[11] TinyOS Community Forum, `http://webs.cs.berkeley.edu/tos`.

[12] H. Woesner, J.-P. Ebert, M. Schläger, and A. Wolisz, “Power-Saving Mechanisms in Emerging Standards for Wireless LANs: The MAC Level Perspective,” *IEEE Personal Communications*, pp. 40–48, June 1998.

[13] E.-S. Jung and N. H. Vaidya, “An Energy Efficient MAC Protocol for Wireless LANs,” in *IEEE Infocom 2002*, June 2002.

[14] C. S. Raghavendra and S. Singh, “PAMAS – Power Aware Multi-Access protocol with Signalling for Ad Hoc Networks,” *ACM Computer Communications Review*, July 1998.

[15] T. van Dam and K. Langendoen, “An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks,” in *ACM SenSys 2003*, November 2003.

[16] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie, “Protocols for Self-Organization of a Wireless Sensor Network,” *IEEE Personal Communications*, vol. 7, no. 5, pp. 16–27, October 2000.

[17] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves, “Energy-Efficient, Collision-Free Medium Access Control for Wireless Sensor Networks,” in *ACM SenSys 2003*, November 2003.

[18] C. R. Lin and M. Gerla, “Real-time Support in Multihop Wireless Networks,” *ACM Wireless Networks*, vol. 5, no. 2, pp. 125–135, March 1999.

[19] M. L. Sichitiu, “Cross-Layer Scheduling for Power Efficiency in Wireless Sensor Networks,” in *IEEE Infocom 2004*, March 2004.

[20] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, “Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks,” in *ACM MobiCom 2001*, July 2001.

[21] Y. Xu, J. Heidemann, and D. Estrin, “Geography-informed Energy Conservation for Ad Hoc Routing,” in *ACM MobiCom 2001*, July 2001.

[22] A. Cerpa and D. Estrin, “ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies,” in *IEEE Infocom 2002*, June 2002.

[23] R. Zheng and R. Kravets, “On-demand Power Management for Ad Hoc Networks,” in *IEEE Infocom 2003*, April 2003.

[24] J. M. Rabaey, M. J. Ammer, J. L. da Silva Jr., D. Patel, and S. Roundy, “PicoRadio Supports Ad Hoc Ultra-Low Power Wireless Networking,” *IEEE Computer*, July 2000.

[25] C. Guo, L. C. Zhong, and J. M. Rabaey, “Low Power Distributed MAC for Ad Hoc Sensor Radio Networks,” in *IEEE GlobeCom 2001*, November 2001.

[26] J. M. Rabaey, J. Ammer, T. Karalar, S. Li, B. Otis, M. Sheets, and T. Tuan, “PicoRadios for Wireless Sensor Networks — The Next Challenge in Ultra-Low Power Design,” in *IEEE International Solid-State Circuits Conference (ISSCC) 2002*, February 2002.

[27] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava, “Topology Management for Sensor Networks: Exploiting Latency and Density,” in *ACM MobiHoc 2002*, June 2002.

[28] ——, “Optimizing Sensor Networks in the Energy-Latency-Density Design Space,” *IEEE Transactions on Mobile Computing*, vol. 1, no. 1, pp. 70–80, January-March 2002.

[29] M. J. Miller and N. H. Vaidya, “Minimizing Energy Consumption in Sensor Networks Using a Wakeup Radio,” in *IEEE WCNC 2004*, March 2004.

[30] M. J. Miller, “Minimizing Energy Consumption in Sensor Networks Using a Wakeup Radio,” Master’s thesis, University of Illinois at Urbana-Champaign, November 2003.

[31] E. Shih, P. Bahl, and M. J. Sinclair, “Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices,” in *ACM MobiCom 2002*, September 2002.

[32] J. Deng and Z. J. Haas, “Dual Busy Tone Multiple Access (DBTMA): A New Medium Access Control for Packet Radio Networks,” in *IEEE International Conference on Universal Personal Communication (ICUPC) 1998*, October 1998.

[33] Z. J. Haas and S. Tabrizi, “On Some Challenges and Design Choices in Ad-hoc Communications,” in *IEEE MILCOM 1998*, October 1998.

[34] W. Stark, “Coding, Modulation, and Multiple-Access,” in *WTEC Study on Wireless Technologies and Information Networks*, 2000, ch. 2, pp. 5–14, `http://www.wtec.org/loyola/wireless/02_03.htm`.

[35] D. B. Johnson and D. A. Maltz, “Dynamic Source Routing in Ad Hoc Wireless Networks,” in *Mobile Computing*, T. Imielinski and H. Korth, Eds. Kluwer Academic Publishers, 1996, ch. 5, pp. 153–181.

[36] C. E. Perkins and E. M. Royer, “Ad-Hoc On Demand Distance Vector Routing,” in *IEEE WMCSA 1999*, February 1999.

[37] W. D. List and N. H. Vaidya, “A Routing Protocol for k-Hop Networks,” in *IEEE WCNC 2004*, March 2004.

[38] B. Awerbuch, D. Holmer, and H. Rubens, “The Pulse Protocol: Energy Efficient Infrastructure Access,” in *IEEE Infocom 2004*, March 2004.